Empirical study on the difficulties of software modeling through class diagrams

Pamela Flores
Escuela Politécnica Nacional
pamela.flores@epn.edu.ec

Carlos Andrés Rodas
Escuela Politécnica Nacional
carlos.rodas@epn.edu.ec

Jenny Torres
Escuela Politécnica Nacional
jenny.torres@epn.edu.ec

Abstract

Software design is one of the stages of the software life cycle characterized as an activity of a creative nature, where software components and their relationships are identified, hence it is extremely important for constructing software efficiently. This research aims to explore the problems students at the undergraduate level face in their first attempts at modeling software. In this article we report the results of an empirical case study that analyzes class diagrams expressed in the Unified Modeling Language (UML) by students enrolled in lectures related to computer science at the undergraduate level. Additionally, we conducted a quantitative analysis that makes evident the most frequent problems the students faced while designing software. The results reveal that students show difficulties understanding multiplicity, assigning appropriate behavior to classes, distinguishing attributes from classes, and conceiving the problem from a holistic perspective.

Keywords:
Object-oriented approach, class diagram, difficulties, empirical study, thematic analysis.

1. Introduction

The Object-Oriented approach (OO) to software design is a creative activity that identifies components and their relationships based on requirements. It involves the design of classes and the relationships between them. These classes define both the objects in the system and their interactions [1]; all kinds of software design scenarios can be expressed through the Unified Modeling Language (UML), as it is a widely accepted design notation used for software development [2]. UML modeling skills are necessary for many subjects related to software engineering and therefore are taught at the university level [3].

The initial teaching of OO design and programming is accompanied by many problems, such as the complexity of the characteristics of the programming language, problems designing software, and the fragile knowledge of beginners [4]. Designing software is a complex activity that involves mental models that students struggle to learn during introductory modeling and design courses. In consequence, OO design and programming can be problematic, since its concepts are closely related to each other and cannot be easily taught and learned in isolation [5].

According to the literature, the most common difficulties that students have while designing software are: lack of understanding of object-oriented concepts [6, 7, 8, 9], confusion between concepts related to object, class, collections and modeling [10], confusion in class-object, attribute-method and class-subclass association including concepts of inheritance [11], syntactic and notation errors while creating UML diagrams [11, 12], and misuse of inheritance, relationships, and function names [6, 13, 14].

In order to contribute to the literature on this topic, an empirical study was carried out to identify difficulties that undergraduate students enrolled in lectures related to computer science had when designing software with the object-oriented approach through class diagram. A quantification was made to show and discuss the most frequent problems.

The rest of the article is structured as follows. Section II shows related work found in literature. Section III presents the research methodology. Section IV shows the development of the qualitative research and quantitative analysis. Section V presents the study findings. Section VI presents the reliability of the research. Finally, Section VII concludes the paper and presents future work.

2. Related Work

There is diverse literature related to the difficulties when designing and teaching object-oriented software. The related work connected to problems of the

URI: https://hdl.handle.net/10125/79457
978-0-9981331-5-7

object-oriented approach can be divided in qualitative studies, and studies with a different approach.

In the research related to qualitative studies, the authors of this research previously conducted a study [6] to explore design decisions under the object-oriented approach. In this work students show various difficulties when constructing class diagrams, the possible causes of these difficulties were identified as: the strict copy of reality when designing classes, the influence of previous experience using the structured approach, the oversimplification of the class diagrams, and the lack of understanding of object-oriented concepts. The research conducted by Ragonis et al. [15] focuses on understanding the learning concepts of object-oriented programming in first-year students through a qualitative longitudinal study. The findings of this research were divided into four main categories: class versus object, instantiation and builders, simple versus composite classes, and program flow. In these four categories, conceptions and difficulties are identified. Sanders et al. [16] conducted a qualitative study of software programs performed by a group of students. The research carried out the analysis of erroneous concepts in their programs, resulting in a classification of errors categorized into mechanics, instance/class conflation, problems with linking and interaction, class/collection conflation, problems with hierarchies, and failures in modeling.

On the other hand, in other studies related to problems designing object-oriented software, Reuter et al. [17] conducted an empirical study applying techniques such as *think aloud* and observing students during the modeling process. The authors generated a catalog of problems related to the different UML diagrams, however, only one problem related to the class diagram is showed, which states that students create independent classes without any relationship between them. The work presented by Ven Yu Sien [18] studies the difficulties and misunderstandings of undergraduate students when modeling software through class and sequence diagrams. Their findings show a lack of identification of related concepts within the domain problem, problems with misassigned or not assigned attributes, and a disconnection between the class diagram and their respective sequence diagram. In conclusion, the students generally produced incomplete or inconsistent models. The research conducted by Pourali et al. [19] presents an empirical study to identify the most prominent difficulties users might face when using modeling tools for developing UML class diagrams. The researchers analyzed Class and State-Machine models developed by students as course assignments. It concludes that the greatest difficulties for users are remembering contextual information, and identifying and fixing errors and inconsistencies. Chren et al. [20] present the common errors of the different types of UML diagrams, and also uses this catalog of errors to analyze projects of software engineering students. This study reports that one of the errors generated in class diagrams is the absence of methods in classes.

Finally, in the work related to difficulties when teaching object-oriented design, Engels et al. [21] analyzes the teaching of different UML diagrams in the context of the software development process, demonstrating that the use of UML is a vehicle for teaching fundamental concepts of software engineering. Silva et al. [14] present another study related to teaching where the objective of the research is focused on understanding how active learning strategies influence the teaching and learning of UML diagrams. The authors conducted a case study with undergraduate students, the findings of which focus mainly on the challenges that instructors face when teaching UML.

This study differs from the works mentioned previously in two ways: (1) it identifies the difficulties of the students specifically in the modeling of a problem through class diagrams, (2) it allows us to know and discuss the frequency of problems that students had when performing software design exercises.

3. Research Methodology

This section shows the research questions, the chosen methodology, and the details of the selected case study.

3.1. Research Questions

This research was carried out trying to answer two questions:

- What are the difficulties that students present when designing through class diagrams?
- What are the most frequent difficulties that students show when designing through class diagrams?

3.2. Research Method

The research questions posed for this study require a qualitative and quantitative data analysis perspective. This research uses qualitative data that includes all nonnumerical data as words, images, sounds, etc. These data can be generated with case studies, action research and ethnography. On the other hand, quantitative

analysis is required to look for patterns in the data and draw conclusions [22].

In this study, we have applied *thematic analysis*, which is defined as a research methodology for identifying, analyzing, organizing, describing, and reporting themes found within a data set. This qualitative method made it possible to identify, analyze and report on the patterns of data [23]. In addition, in order to know the frequency of problems in students, it was necessary to make a quantitative analysis. The analysis of quantitative data involves a process of abstraction that starts from the qualitative data research, previously analyzed [22].

3.3. Setting

The case study was conducted with a group of students at University "H" in the Faculty of Informatics. The details are shown below.

Subject

The subject is *Modeling and Software Design*, a compulsory lecture of the fifth semester in the Faculty of Informatics. It is taught 6 hours per week for 16 weeks. Students in this class must have previously completed *Database Management, Programming I, and Programming II*; in addition, while students take *Modeling and Software Design*, they also take *Software Engineering I* in parallel. The content of *Programming II* focuses on the teaching of object-oriented programming languages, so that students already have previous knowledge about the concepts of this approach at the programming level. This course focuses primarily on software design and modeling at a detailed level, with an emphasis on class diagrams, for this reason, this study analyzes the class diagrams made by the students.

Lecture structure

There are 16 weeks during the semester. The content of the lectures is as follows: 1. Software Design and Process Modeling, 2. Software Development Paradigms, 3. Decomposition in software design, 7. Abstraction in software design, 8. Information Hiding Principle, 9. Design Patterns and Observation Pattern, 10. Facade and MVC pattern, 11. Decorator and Factory Pattern, 12. State pattern, 13. Chain responsibility pattern.

Participants

The group of students in *Modeling and Software Design* during this study were 29, all of them participated in the study.

Test

The test was conducted in the seventh week of the semester and consisted of three exercises. The exercises presented to the students for this case study were chosen because they allow the application of the concepts learned in the academic period. Exercise *Betting* involves the use of inheritance and decomposition. The exercise *Circle* is related to graphical objects that allows to know how the students conceive the problem. Finally, the exercise *Hotel* is a transactional exercise whose characteristics allow to know the understanding of the objects in an exercise that can be solved in a structured way. It is important to notice that the time allocated for each exercise, and its complexity, have not been considered for this study, as suggested in the work of Kuutila et al. [24]; but they should be considered in future research. The statement of each exercise is shown below.

- Exercise Betting

An application is required to take over the betting service, where the user must register in the system to have an account to manage their bets. Bets can be accepted by transfer or by card. The system supports different types of bets, for example single bet (choosing the winner team), special bet (choosing the minute when the first goal is scored) and others.

- Exercise Circle

This application consists on drawing a small circle inside a larger circle. The smaller circle can move inside the larger circle, without getting out of it.

- Exercise Hotel

This application is responsible for booking rooms in a hotel. It is necessary to take into account the booking dates and the verification of room availability.

4. Analysis process

This study was conducted under two approaches: qualitative and quantitative. A qualitative case study has been carried out, through the analysis of the exercises accomplished by the students, while the quantitative approach allowed us to do a quantitative analysis to look for patterns in the data.

4.1. Qualitative data analysis

In this section we describe the main components that make up the qualitative analysis.

Qualitative data collection

The first step was to collect qualitative data. In this study the main analysed document was the test applied to the students, this test consisted of three exercises, the same ones mentioned in Section 3.3.

Qualitative data encoding

At this stage a deductive coding was carried out. For this, we have taken as a reference the problems that were previously identified in a work published by the same authors of this study [6], however, it is important to mention that some of the findings shown in this study have also been reported by other authors [16, 18, 19, 25]. In the previous study ten student's design decisions were identified, which are shown below:

- Tendency to create a third class between two other classes to associate them, instead of creating a many-many association between them.
- Assigning the behavior of a real-life object to the class diagram as is, instead of using an abstraction of that concept at the software level.
- Lack of creation of classes for relevant aspects of the application.
- Designing classes with different names, but with the same structure.
- Designing classes without any behavior. Special interest in defining classes only through their attributes.
- Place responsibilities on classes that should not be responsible for that behavior.
- Creating classes that differ from their superclass or sibling classes only by its attribute values, when the behavior should be the same.
- Assignment of complex behaviors as attributes.
- Belief of students that placing an ID attribute in each class will allow them to access all instances of that class.
- Definition of classes that are not concepts.

It is important to mention that in this study we did not limit ourselves to coding only the problems mentioned above. The experts were free to code problems that were not taken into account in the previous study [6]. Consequently, in the present research we found 8 of the 10 problems previously identified and 7 additional problems, so that the 15 problems found are explained later. At this stage a total of 365 codes were obtained, the coding was performed separately by two experts, to then apply the *peer evaluation technique*, in order to guarantee the coding process. The exercises were coded with the software ATLAS.ti [26].

Qualitative data refinement

All code obtained in the previous phase were reviewed one by one and subsequently matched or separated. After this phase, the resulting refined codes were 266.

Grouping of qualitative data

The 266 refined codes were grouped into fifteen categories. The details of the resulting categories, the examples and their respective acronyms are shown below:

1. Convert attributes into classes (CLA), refers to extract an attribute and convert it into a class. However, this created a class that represents only data, without behavior. For example when the student designs a `Circle` class and a `Position` class, but the latter has only attributes `Xo` and `Yo`.
2. Not considering the problem from an holistic perspective (HOL). This category is related to the fact that students do not conceive all the aspects necessary to solve the problem. For example, in the exercise *Betting*, the student should check all aspects that influence the resolution of the exercise, such as whether there are sufficient funds for a person to place a bet.
3. Not including the classes necessary for the design (NUM), refers to omitting classes in the diagram in spite of being explicitly mentioned in the statement, for example, the absence of the `Account` class in the exercise *Betting*.
4. Creation of classes that should be related to a concept (FUN), but the concept itself does not exist in the diagram. For example, when a student has created a class named `BetType`, but the concept `Bet` does not exist in the diagram.
5. Incorrect use of multiplicity between classes (LIS) because the student does not identify the possible existence of several instances of the same class. For example, in the exercise *Hotel*, some students did not identify that multiple reservations can be made to the same room.
6. Classes with inadequate or insufficient behavior (COM), this category refers to those classes that were created with a behavior foreign to the concept of the class or the behavior only partially represents the concept, for example, the creation

of a `Hotel` class, which has `reserveRoom()` method.

7. Creating the same class multiple times on a single class diagram (INS), instead of instancing the class multiple times. For example, the creation of different rooms, such as `SingleRoom`, `DoubleRoom`, `TripleRoom`, instead of just the class `Room`.
8. Defining attributes that could be a class (ATR), an example of this is when the student creates a `Reserve` class, and places an attribute `roomNumber`, and there is no `Room` class in the design.
9. Placement of different methods that could have been represented by a single method (MET), for example, when it is placed `move-left`, and `move-right` instead of `move`.
10. Classes built in the image and likeness of concepts of the real life (REA), for example, in the case of a student who created a `Room` class with a `toClean()` method.
11. Creating a `Main` class, which only function should be to trigger the start of the program, and filling it with functions that should be part of other classes (MAI).
12. Creation of classes whose name and behavior represent an action and not a concept (ACC), an example was presented in the exercise *Circle*, where a student diagrammed a class labeled `SmallCircle`, and additionally two classes: one labeled `Draw` and one `Move`.
13. Creating relationships between confusing or erroneous classes (PCL), it refers to syntax errors in UML semantics. For example, using the composition relationship instead of aggregation.
14. Construction of classes with attributes but no methods, even when they needed methods with distinct behaviours in the context of the exercise (SIC). For example, the `Client` class with the attributes `name`, `lastName`, `id`, and without any methods.
15. Construction of inheritance structures whose derived classes only differ from the base class by their attributes (HER). For example, when students created a base class called `Bet` with an implemented `bet()` method, and two subclasses, one called `IndividualBet` with an attribute called `individualFactor` and the other `ComplexBet` with an attribute called `complexFactor` but both with the same implemented method inherited from the base class called `Bet`

4.2. Quantitative data analysis

This subsection presents the detail of the quantitative analysis with respect to the frequency of design problems in each category and in each exercise.

Frequency of problems per student

This section details the number of appearances of design problems of each of the students in the three test exercises, it is divided by category.

As an example, student 7 has been chosen to explain the analysis that was carried out with all the students. In Figure 1 we present the resulting graph of student 7, which shows on the *x-axis* the categories generated in the previous qualitative analysis and on the *y-axis* the number of occurrences of problems in each exercise. For this, the exercises in the graph have been differentiated by colors with blue color the exercise *Betting*, with yellow color the exercise *Circle* and with green color the exercise *Hotel*.

In Figure 1, student 7 has two design problems in the category labeled ATR in the exercise *Betting*. He also has two problems in the LIS category in the *Hotel* and *Betting* exercises respectively. Additionally, student 7 has one problem in the HOL, NUM, COM, MAIN and PCL categories.

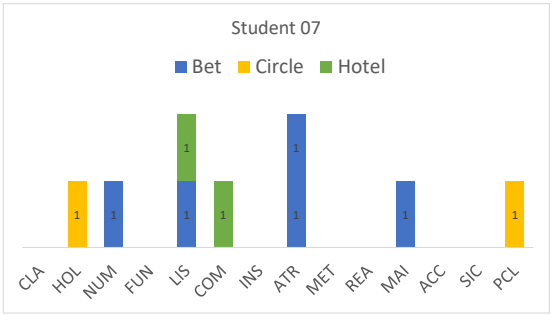


Figure 1. Number of problems per category in each exercise of student 07

This analysis was performed with each of the 29 participants in the study, thus obtaining the categories where the highest number of occurrences of design problems were recorded, which are: ATR, LIS, COM, and HOL as shown in Figure 2.

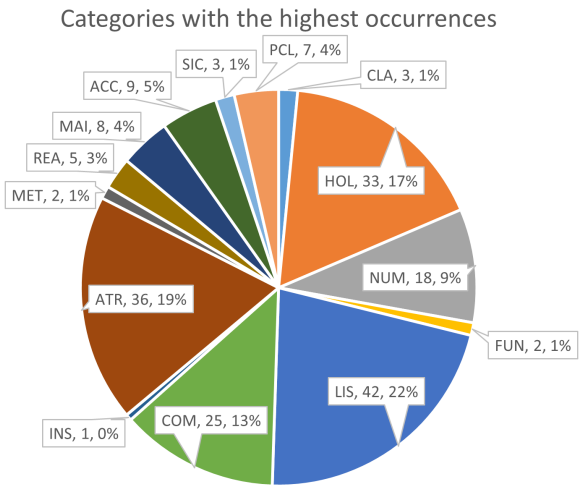
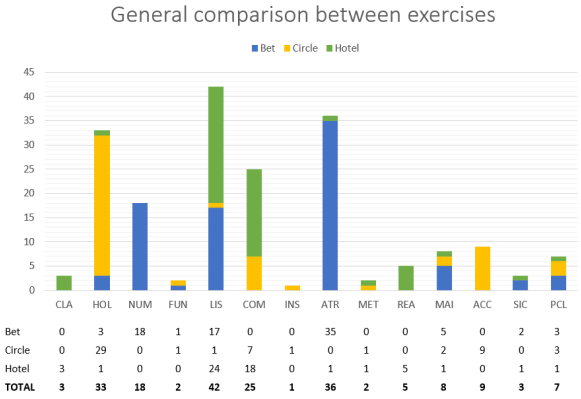


Figure 2. Categories with the highest occurrences

Frequency of design problems by category and by exercise

Figure 3 shows a compendium of problems by category for each of the three exercises. The category with the highest frequency of problems is LIS, followed by ATR and HOL. The category with the fewest problems is INS.



	CLA	HOL	NUM	FUN	LIS	COM	INS	ATR	MET	REA	MAI	ACC	SIC	PCL
Bet	0	3	18	1	17	0	0	35	0	0	5	0	2	3
Circle	0	29	0	1	1	7	1	0	1	0	2	9	0	3
Hotel	3	1	0	0	24	18	0	1	1	5	1	0	1	1
TOTAL	**3**	**33**	**18**	**2**	**42**	**25**	**1**	**36**	**2**	**5**	**8**	**9**	**3**	**7**

Figure 3. General comparison between exercises

5. Findings

Answering the first research question: What difficulties do students show when designing through class diagram? We obtained 15 categories that encompass the problems that arose at the time of the design and that were present in the 3 exercises analyzed. The problems of major occurrence in the students, allowed us to answer the second research question. From the study presented in Section 4.2, it can be summarized that the categories where students have the highest number of problems are, in order of major to minor, the categories LIS, ATR, HOL and COM. On the other hand, from the 29 participants, 12 students had problems in LIS and 12 in ATR. When the diagnosis is made by exercise, in exercise *Betting*, the categories with the most problems are ATR and LIS, in exercise *Circle*, the category is HOL and in exercise *Hotel*, ATR and LIS are the categories with the most problems. In this sense, the categories with the most appearances of problems are LIS, COM, ATR and HOL. The implications of each of these four categories are discussed in more detail below.

5.1. Incorrect use of multiplicity between classes (LIS)

Class diagrams allow us to show the classes and the associations between them. Additionally it allows us to visualize the number of objects involved in the association through multiplicity. Thanks to the multiplicity it is possible to define an exact number of objects that are involved; or, if * is used, it indicates that there are an indefinite number of objects in the association [1]. In this way, UML allows to specify the role of the objects that participate in the association.

In this study there were manifestations related to the LIS category in the exercise *Betting*. One of the expected multiplicities was between the `Bet` and `Gambler` classes, since the person making the bet could place several bets, and this was not considered by many students. Most of the students who made the `Bet` and `Gambler` classes performed a multiplicity of 1 to 1 instead of 1 to *. This is evidenced when students did not draw any multiplicity or when they wrote methods such as `getAllBet()` in the class `Bet`, without knowing where or how they handle all bets.

At the software design level, another relationship is aggregation relationship which is used between two classes and is a type of association, which means that an object (the whole) is formed by other objects (the parts)[1]. It is required to define this multiplicity when you want to express the existence of more than one object of the same type, you can also use aggregation to represent a physical container.

Students do not abstract globally, usually thinking that an object has a specific task. When students realize that the task is to manage a set of objects, they understand the need for some mechanism to deal with multiple instances; however, they are unable to define multiplicity correctly. The difficulty is also related to the conception that to a whole and its parts is not always container-like, rather this whole/parts relationship is

more conceptual [27].

The difficulty of define the multiplicity is a persistent problem that has been manifested in several nuances. Being a possible cause of this problem, the difference in between structured and object-oriented approach, where conceptually there is no data and all elements are variables. This possible cause lies largely in a lack of understanding of the object concept rather than in a direct relation to problems with the UML.

LIS is the category that had the highest number of occurrences of problems in the *Bet* and *Hotel* exercises, however in exercise *Circle* it did not have many appearances, because in that exercise it was not required to use multiple objects for its resolution, unlike the first exercises.

5.2. Classes with inappropriate or insufficient behavior (COM)

The concepts of a class and an object are very narrow. However an object is a concrete entity that exists in time and space, while a class represents only an abstraction [27]. That is why abstraction is a fundamental concept in the object-oriented approach. When defining a real life object as a class, with its attributes and relevant methods, it is a necessity to use abstraction to reduce the object to only the parts that are needed for the software that is being designed [28]. In this sense students in this study have difficulties in giving the class the right behavior and this has been represented in different ways. Sometimes because methods associated with the class do not correspond to the concept that this class represents, or because there is an overload of methods with low cohesiveness between them, or there is a class without a behavior.

An example of the COM category was seen in the exercise *Hotel*, when the `Room` class has a method `moveFurniture()`, or in the exercise *Betting* where students assigned to the `Bet` class behaviors related to the verification of aspects of the event.

We have also seen classes with an overload of methods with little cohesion between them, for example, a `Bet` class with methods related to the payment and the registration of the gambler. Although the `Bet` class at first glance has one "behavior". The `Bet` class is a clear example of an overloaded class that does many different things. The overloading of methods in a class has also been cited in other works [29, 30]. We also find classes defined only with attributes, such as `Client` class and `Hotel` class, or absence of methods in classes [20].

Many authors describe this problem when defining classes, some of them attribute it to the confusing behavior of assigning a "real" behavior of the physical object to the software object. This was also a finding reported in [15], who conducted a study where students were asked to create a composite class consisting of several simple classes, where the composite class was called `Room` and the simple classes: `Mirror`, `Bed`, and `Cupboard`. The students placed the `addMirror` method in the `Room` class. The authors interpret this behavior as a student confusion, since it is a possible situation in real life. This involves assigning the erroneous behavior to the `Room` class; related results were also reported by [6].

Other studies conducted by [10], report difficulties of students in conceiving a class as an abstraction of some kind of entity in the real-world problem domain. Although some authors defend the idea that objects have the property of naturalness, which is understood as the property that allows mapping the physical objects of the problem domain to the software. [31, 32].

Also some students have created classes built only with the `get()` and `set()` methods, giving the false sensation that these have behavior, when these methods indicate that through them the attributes of that class can be accessed from outside, rather than the behavior of the class itself. Students are often motivated to use `get()` and `set` methods to hide the modules, being a misinterpretation of the Information Hiding Principle [33]. The difficulty of defining objects has also been documented in the literature [34, 32, 28].

5.3. Defining attributes that could be a class (ATR)

This category refers to the simplification of a concept by defining it as an attribute of a class, instead of having conceived it as a class by itself due to its complexity. Some students believe that placing "few attributes" is a way to define correctly a concept. They showed this behavior, when they placed in the `Circle` class an attribute called `type` and in `Bet` class an attribute called `typeOfBet`.

Furthermore, difficulties related to misassigned attributes and missing attributes have also been found in the literature [18]. However, there is an important tendency to think that a concept can be defined only with attributes, leaving aside methods. This is also related to the behavior to which the structured approach has accustomed us, where data are the food for functions, as [35] defines systems under the structured approach: *"A software system is a system that manipulates and stores data"*, so that data under this approach have a leading role. The influence of the structured approach on the implementation of the object-oriented approach has already been discussed in the literature

[6]. However, their manifestations go beyond giving more relevance to the attributes. These results coincide with those analyzed in [28], where students assigned to the `Employee` class the methods to calculate the salary of an employee, when these should belong to the `Human Resources` class. This behavior shows a clear procedural design where the `Employee` class is in control and the `Human Resources` class is just a data. Detienne [36, 37] also shows his findings on the process to which novice learners converge when they decompose large procedures into smaller functional units, reflecting the tendency to associate in a single class the procedure as a whole. Finally, the work presented by Ven Yu Sien [18] in their findings show a lack of identification of related concepts within the domain problem and problems with misassigned or not assigned attributes.

As in the previous category, students show a clear lack of abstraction by not being able to conceptualize a concept through a class with its own behavior or by reducing a concept to an attribute.

5.4. Not considering the problem from a holistic perspective (HOL)

This category is related to the fact that the students do not conceive all the aspects necessary for the resolution of the problem. This finding could be interpreted in different ways, however, it can be greatly influenced by the fact of how to understand the exercise, since it can be deduced that the student focused only on the main task.

In this study, HOL was the category with the most frequent appearance of design problems in the exercise *Circle*, most of the difficulties were related to not considering aspects such as collision between circles in the exercise related to moving a small circle inside a larger one. There is no evidence that this category is related to abstraction ability of the students.

Novice programmers have great difficulty making good design decisions because they violate the principles of the object-oriented approach resulting in poor quality designs and coding [38, 39, 10]. Although there are guidelines and recommendations for object-oriented software design, novice modelers usually do not apply them due to lack of understanding of its principles [6, 7].

It is important to mention that although the less frequent categories have not been discussed in depth, there is literature that reports the same results. For example, the NUM category, related to the omission of classes in the diagram, has also been reported in [18], where the participants of the study demonstrated not having identified the expected classes. In this same study, problems were also evidenced in the semantics of the aggregation and composition relationships, which could have been categorized in the PCL category of this study. Also, [19] shows the difficulties related with remembering the contextual information and identifying the whole problem.

6. Threats to reliability

Qualitative research has been widely criticized for not providing enough information about the analysis of the data and how it has worked from the raw data to its conclusions. This study adheres to the quality criteria set forth by Yvonna S. Lincoln, Egon G. Guba [40], W. Lawrence Neuman [41] and Sharan B. Merriam [42] in the educational context.

On one hand, the work has *reliability*, that is, the consistency of the results obtained from the data. To ensure reliability, the researchers of this study, instead of requiring that people outside the research agree that, based on the data collected, the results make sense, are consistent and reliable. They detailed the trace-ability of the source data and the decisions taken to reach their conclusions. The details of the environment and participants are also described, which will allow other researchers to apply this study in similar contexts.

On the other hand, *validity* that means truthfulness, but in the qualitative context we could rather speak of authenticity, which means capturing a detailed view of the research process. To ensure validity in this research, we have applied strategies such as triangulation, by using several researchers so that each exercise were analyzed and coded separately by the researchers, so that the codes and categories were consensual through peer debriefing techniques, ensuring the credibility of the investigation in this way.

In the presented research students go through different stages of learning: a) when the concepts are presented to the students, b) when they do exercises to try to learn the concepts, and c) when the students take the assessments. In this sense, it should be noted that there is a possible threat to the validity of the research because the stage in which the students present the problems was not identified, nor were the causes of the problems. It is important to recognize that the problems might have been caused by the approach of the teacher while teaching the topic rather than the approach of the students while learning it. Nevertheless, neither the identification of the stage nor the causes were considered within the scope of the study.

In addition, to avoid ethical conflicts regarding the manipulation of the data collected from the students,

informed consent forms were prepared to guarantee anonymity and confidentiality of the data obtained from the students. This report was read and signed by the students before the research.

7. Conclusions and Future Work

The research questions posed for this study have been answered. Our study has revealed fifteen categories representing common difficulties of students when designing. The most frequent categories were those related to incorrect use of multiplicity between classes (LIS). Additionally, the most frequent categories were explained through the literature.

Based on our analysis, we concluded that students have several drawbacks when assigning multiplicity between classes, with this category (LIS) having the most occurrences of design problems. It is followed by the category indicating that students define classes with behaviors not related to their concept (COM), for example, classes without methods, classes only with `get()` and `set()` methods or classes with methods that do not match. Another category that stood out among the students was the one related to the placement of an attribute that due to its complexity could be a class (ATR). Finally, it was also evident the category which shows that some students do not conceive the problem in a comprehensive manner, as they do not consider all the aspects required to solve the problem (HOL).

Based on the literature, the misuse of multiplicity in a design may be caused by lack of clarity in the handling of a set of objects, while the assignment of inappropriate or insufficient behavior to the classes reveals that students have difficulties in conceiving the essence of a concept, where a possible cause is the strong influence of copying the behavior of a real life object strictly. As for the assignment of attributes that could be classes, could be related to not being able to identify an attribute with a concept that could have behavior. Finally, the difficulty related with not considering the problem from an holistic perspective, could be founded in the great difficulty in adopt the principles and concepts of the object-oriented approach.

These results allow software design teachers to better understand and pay attention to the difficulties students have when designing software. This work raises another question to examine further: Do the most frequent problems persist after training? In this sense, we will analyze the same exercises after training, to know the persistent ideas around software design.

Acknowledgment

This research was supported by *Escuela Politécnica Nacional del Ecuador* through the grant research project PIS-19-11. In addition, we acknowledge and thank the Anonymous Reviewers for their valuable recommendations, which contributed to improving the quality of this paper.

References

[1] I. Sommerville, "Software engineering 9th edition," *ISBN-10*, vol. 137035152, p. 18, 2011.

[2] W. A. F. Silva, I. F. Steinmacher, and T. U. Conte, "Is it better to learn from problems or erroneous examples?," in *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*, pp. 222–231, IEEE, 2017.

[3] S. Frezza and W. Andersen, "Interactive exercises to support effective learning of uml structural modeling," in *Proceedings. Frontiers in Education. 36th Annual Conference*, pp. 7–12, IEEE, 2006.

[4] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion," *Computer science education*, vol. 13, no. 2, pp. 137–172, 2003.

[5] A. L. Santos, "Aguia/j: A tool for interactive experimentation of objects," in *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, (New York, NY, USA), p. 43–47, Association for Computing Machinery, 2011.

[6] P. Flores, J. Torres, and R. Fonseca-Delgado, "Design decisions under object-oriented approach: A thematic analysis from the abstraction point of view," in *Proceedings of the 8th Computer Science Education Research Conference*, pp. 89–97, 2019.

[7] Z. Ma, "An approach to improve the quality of object-oriented models from novice modelers through project practice," *Frontiers of Computer Science*, vol. 11, no. 3, pp. 485–498, 2017.

[8] D. P. Tegarden and S. D. Sheetz, "Cognitive activities in oo development," *International Journal of Human-Computer Studies*, vol. 54, no. 6, pp. 779–798, 2001.

[9] T. Gorschek, E. Tempero, and L. Angelis, "A large-scale empirical study of practitioners' use of object-oriented concepts," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pp. 115–124, 2010.

[10] S. Xinogalos, "Object-oriented design and programming: an investigation of novices' conceptions on objects and classes," *ACM Transactions on Computing Education (TOCE)*, vol. 15, no. 3, pp. 1–21, 2015.

[11] P. Hubwieser and A. Mühling, "What students (should) know about object oriented programming," in *Proceedings of the seventh international workshop on Computing education research*, pp. 77–84, 2011.

[12] M. Kayama, S. Ogata, K. Masymoto, M. Hashimoto, and M. Otani, "A practical conceptual modeling teaching method based on quantitative error analyses for novices

learning to create error-free simple class diagrams," in *2014 IIAI 3rd International Conference on Advanced Applied Informatics*, pp. 616–622, IEEE, 2014.

[13] S. Frezza and W. Andersen, "Interactive exercises to support effective learning of uml structural modeling," in *Proceedings. Frontiers in Education. 36th Annual Conference*, pp. 7–12, IEEE, 2006.

[14] W. Silva, I. Steinmacher, and T. Conte, "Students' and instructors' perceptions of five different active learning strategies used to teach software modeling," *IEEE Access*, vol. 7, pp. 184063–184077, 2019.

[15] N. Ragonis and M. Ben-Ari, "A long-term investigation of the comprehension of oop concepts by novices," *Computer Science Education*, vol. 15, no. 3, pp. 203–221, 2005.

[16] K. Sanders and L. Thomas, "Checklists for grading object-oriented cs1 programs: Concepts and misconceptions," in *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '07, (New York, NY, USA), p. 166–170, Association for Computing Machinery, 2007.

[17] R. Reuter, T. Stark, Y. Sedelmaier, D. Landes, J. Mottok, and C. Wolff, "Insights in students' problems during uml modeling," in *2020 IEEE Global Engineering Education Conference (EDUCON)*, pp. 592–600, IEEE, 2020.

[18] V. Y. Sien, "An investigation of difficulties experienced by students developing unified modelling language (uml) class and sequence diagrams," *Computer Science Education*, vol. 21, no. 4, pp. 317–342, 2011.

[19] P. Pourali and J. M. Atlee, "An empirical investigation to understand the difficulties and challenges of software modellers when using modelling tools," in *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS '18, (New York, NY, USA), p. 224–234, Association for Computing Machinery, 2018.

[20] S. Chren, B. Buhnova, M. Macak, L. Daubner, and B. Rossi, "Mistakes in uml diagrams: analysis of student projects in a software engineering course," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pp. 100–109, IEEE, 2019.

[21] G. Engels, J. H. Hausmann, M. Lohmann, and S. Sauer, "Teaching uml is teaching software engineering is teaching abstraction," in *International Conference on Model Driven Engineering Languages and Systems*, pp. 306–319, Springer, 2005.

[22] B. J. Oates, *Researching Information Systems and Computing*. Sage Publications Ltd., 2006.

[23] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006.

[24] M. Kuutila, M. Mäntylä, U. Farooq, and M. Claes, "Time pressure in software engineering: A systematic review," *Information and Software Technology*, vol. 121, p. 106257, 2020.

[25] D. R. Stikkolorum, F. G. de Oliveira Neto, and M. R. V. Chaudron, "Evaluating didactic approaches used by teaching assistants for software analysis and design using uml," in *Proceedings of the 3rd European Conference of Software Engineering Education*, ECSEE'18, (New York, NY, USA), p. 122–131, Association for Computing Machinery, 2018.

[26] S. Friese, "Atlas.ti 8 user manual," Aug. 2018. Accessed: 2021-09-01.

[27] G. Booch, R. Maksimchuk, M. Engle, B. Young, J. Conallen, and K. Houston, *Object-Oriented Analysis and Design with Applications, Third Edition*. Addison-Wesley Professional, third ed., 2007.

[28] R. Or-Bach and I. Lavy, "Cognitive activities of abstraction in object orientation: an empirical study," *ACM SIGCSE Bulletin*, vol. 36, no. 2, pp. 82–86, 2004.

[29] P. Flores, N. M. Martínez, and S. P. Roche, "Persistent ideas in a software design course: a qualitative case study," *The International journal of engineering education*, vol. 32, no. 2, pp. 937–947, 2016.

[30] M. P. Monteiro, "On the cognitive foundations of modularity.," in *PPIG*, p. 22, 2011.

[31] G. White and M. Sivitanides, "Cognitive differences between procedural programming and object oriented programming," *Information Technology and management*, vol. 6, no. 4, pp. 333–350, 2005.

[32] F. Détienne, "Assessing the cognitive consequences of the object-oriented approach: A survey of empirical research on object-oriented design by individuals and teams," *Interacting with computers*, vol. 9, no. 1, pp. 47–72, 1997.

[33] P. Flores, N. Medinilla, and S. Pamplona, "What do software design students understand about information hiding?: A qualitative case study," in *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*, Koli Calling '14, (New York, NY, USA), pp. 61–70, ACM, 2014.

[34] D. Svetinovic, D. M. Berry, and M. Godfrey, "Concept identification in object-oriented domain analysis: Why some students just don't get it," in *13th IEEE International Conference on Requirements Engineering (RE'05)*, pp. 189–198, IEEE, 2005.

[35] R. Wieringa, "A survey of structured and object-oriented software specification methods and techniques," *ACM Computing Surveys (CSUR)*, vol. 30, no. 4, pp. 459–527, 1998.

[36] F. Détienne, "Design strategies and knowledge in object-oriented programming: Effects of experience," *Human–Computer Interaction*, vol. 10, p. 129–169, Sept. 1995.

[37] F. Détienne, *Software Design–Cognitive Aspect*. Springer Science & Business Media, 2001.

[38] F. Steimann, "Fatal abstraction," in *Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pp. 125–130, 2018.

[39] V. Thurner, "Fostering the comprehension of the object-oriented programming paradigm by a virtual lab exercise," in *2019 5th Experiment International Conference (exp. at'19)*, pp. 137–142, IEEE, 2019.

[40] Y. Lincoln, E. Guba, and S. Publishing, *Naturalistic Inquiry*. SAGE Publications, 1985.

[41] W. Neuman, *Social Research Methods: Qualitative and Quantitative Approaches*. Pearson Education, Limited, 2014.

[42] S. B. Merriam, *Qualitative Research and Case Study Applications in Education. Revised and Expanded from" Case Study Research in Education."*. ERIC, 1998.